**University of Toronto**

**ECE450S Software Engineering II** **Spring 2005**

# Project Software – OmniEditor for Graphs

## 1 Introduction and Motivation

OmniEditor was a pilot course project (CSC408H1F 2004/2005) to bridge the gap between different text editors, such as VIM/NetBeans (developed in C/Java), Eclipse (developed in Java/SWT), JEdit (developed in Java/Swing), etc. It relies on Web Services to provide interoperability and reusability. The motivation for the development was stated as follows:
*“ Text editors are the best friends of programmers. There are, however, multiple choices of editors that may confuse a new programmer once he or she is used to one of them. It wastes the programmer’s time to study a new set of key bindings as the time can be spent on more creative tasks. . . . They have different key bindings that lock the programmer in the pro/con camps. As software engineers, we want to bridge the gap between the different editors and help the programmer to use any text editor through familiar key bindings.*
As a continution of the pilot projects, the text-based OmniEditors could be extended to deal with graphics, since the same incompatibility problem exists in the interoperation of graph editors. Our Guinea Pig graph editors is a tool called ”OpenOME” (Java), which can model goal-oriented requirements in software development.
The project involves software reengineering, testing and maintenance. The spirit of team work is very important to the success of software engineering.

## 2 Requirements

The objective of the project is to bridge the OpenOME editors through Web Services. For example, as a user adds/deletes an element in the graph, such change should be seen by other users through the Web.
It is not required to reinvent the wheel, i.e., only need to use one legacy OpenOME editor and the OmniEditor. The students are not recommended to spend lots of time to write another graphic editor or a completely new web service. Instead should understand the existing ones, and extend them.

### Functional requirements

To be more specific, the end user of the target system should be able to edit in a local goal graph while invoking the graph editing services provided by at least one remote graph editor to perform some remote tasks.
The aim of this project is to find a subset of the smallest common divisors of editors such that together, the editors can fulfill useful editing tasks, such as

1. UPLOAD: inform the remote service to open a shared channel with the editing state, such as the content of the local graph and the selection in the buffer;
2. DOWNLOAD: query the remote service about the editing state of potentially other editors, such as the content of a certain channel and a certain selection; [1]
3. INSERT: insert an element into the shared buffer at a certain location;
4. SELECT: select a region of elements;
5. DELETE: delete the selected elements from the shared buffer;
6. COPY: copy the selected elements into the shared buffer;
7. PASTE: copy the shared buffer into the editing area;

Features 1-7 are basic editing operations that are required to be implemented as a web service such that they can be programmably invoked by any remote programs. Each team should finish at least $\{1, 2, 3, 4, 5\}$ operations, $\{6, 7\}$ are optional operations for advanced editings.
Correctness and reliability must be tested for the selected features. Interoperability and extensibility are required too. Responsiveness and usability are desired, but not required.

## 3 Deliverables

### Phase A deliverables

Phase A involves requirement engineering and design recovery. The deliverable should be a document covers requirement specification, feasibility analysis, alternative selections and risk analysis, architecture design and test plan.

### Phase B, C deliverables

Phase B involves the development of the followings.

1. Design the OmniGraphEditor web service that can be used by OpenOME. Each team must use Java programming language to design a web service based on Eclipse. Specify your web service using a WSDL description.
2. Test your own web service and submit your test cases and their verifiable scripts;
3. Publish your web service with its description to the classmates (we will announce the official URL for each team).

[1]We may call the ”UPLOAD/DOWNLOAD” operations ”LOAD/SAVE” from the perspective of the service provider.

The deliverable for Phase B includes the design documents and test cases. The changes must be managed and the released functionality must be unit tested.
Phase C involves an invocation of the web services developed by other teams from the local editor developed by the home team. Each team is now responsible to integrate at least one service and two clients such that they can be used to do useful synchronized tasks (detail requirements will be announced later). The team must study the other's document and validate the other's services. The communication across teams must settle the defects and proceed. Each team must maintain the developed web-service too.

## References

`http://www.cs.toronto.edu/˜yijun/csc408h`
`http://skywolf.cdf.toronto.edu:8081/axis/index.html`
`http://www.cs.toronto.edu/˜yijun/OpenOME.html/`